School of Computation, Information and Technology
Technische Universität München

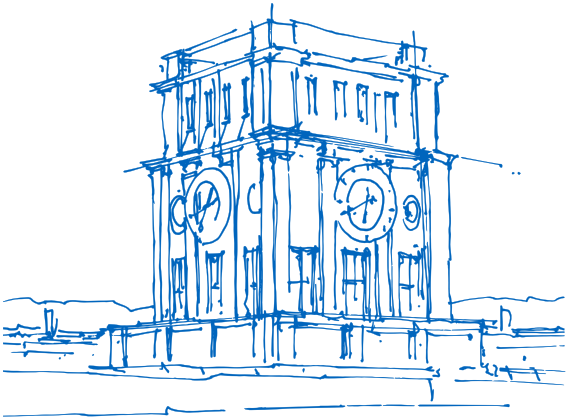
Übung 02: RISC-V Assembly

Einführung in die Rechnerarchitektur

Michael Morandell

Technische Universität München

27. Oktober 2024



Keine Garantie für die Richtigkeit der Tutorfolien.

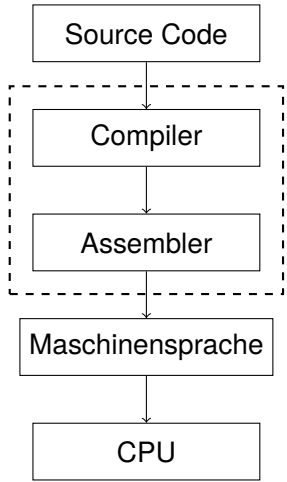
Bei Unklarheiten/Unstimmigkeiten haben VL/ZÜ-Folien recht!

Inhaltsübersicht

- Quiz
- Kurze Wiederholung
- Tutorblatt
  - RISC-V Simulator Einrichtung
  - Registerbenutzung
  - Einfache Befehle
  - Bitmasken und -hacks
  - Vorüberlegungen zur Hausaufgabe

Abstraktionsebenen

- Code in einer Hochsprache (C, Java, . . .) ist lediglich eine Abstraktion
- Compiler: Hochsprache $\rightarrow$ Assemblersprache
- Assembler: Assemblercode $\rightarrow$ Maschinensprache (1:1 Übersetzung)
- Maschinensprache ist plattformspezifisch!
- ISA: „Bedienungsanleitung“ einer CPU
- RISC vs. CISC



RISC-V

- eine von vielen Assemblersprachen
- Datenwortbreite: 32 Bit (4 Byte)
- Little-Endian-Architektur
- 32 Register, einige davon mit spezieller Funktion
- grundlegende Instruktionen auf 32 Bit begrenzt → Konstanten müssen zusammengebastelt werden

Name	Register Number	Use
zero	x0	Constant value 0
ra	x1	Return address
sp	x2	Stack pointer
gp	x3	Global pointer
tp	x4	Thread pointer
t0–2	x5–7	Temporary registers
s0/fp	x8	Saved register/Frame pointer
s1	x9	Saved register
a0–1	x10–11	Function arguments/Return values
a2–7	x12–17	Function arguments
s2–11	x18–27	Saved registers
t3–6	x28–31	Temporary registers

Addition

- addi kann zum Laden von Konstanten verwendet werden
- "Nur"12-Bit Immediate
- Sign extension

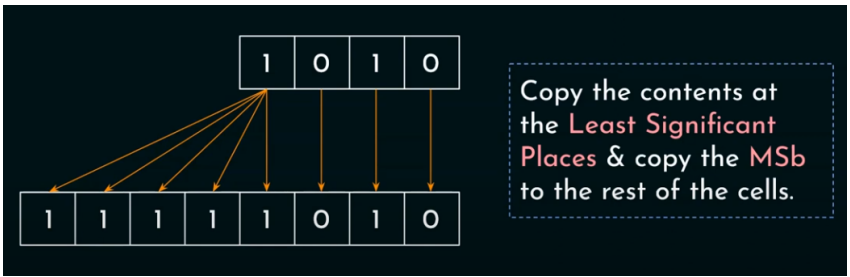
op	funct3	funct7	Type	Instruction	Description	Operation
0010011 (19)	000	–	I	`addi rd, rs1, imm`	add immediate	`rd = rs1 + SignExt(imm)`
0110011 (51)	000	0000000	R	`add rd, rs1, rs2`	add	`rd = rs1 + rs2`
0110011 (51)	000	0100000	R	`sub rd, rs1, rs2`	sub	`rd = rs1 − rs2`

Laden von großen Konstanten

- lui, rd, upimm
  - upimm hat 20 Bit
  - upimm wird zuerst um 12 nach links geshifted
  - dann die unteren 12 Bit mit Null überschrieben
  - anschließend Ergebnis nach rd geschrieben
- Zum Laden von großen Konstanten zuerst lui, dann addi
- Achtung: Sign-Extension von addi kann zu unerwarteten Ergebnissen führen
- Pseudobefehl: li

Sign extension

- 12-Bit Konstante wird auf 32-Bit vergrößert, da Hardware nur mit 32-Bit Zahlen rechnet
- Die spezifizierten 12-Bit werden kopiert und die restlichen Stellen mit dem MSB (Sign Bit) aufgefüllt
- $\rightarrow$ Auffüllen mit 0, wenn Zahl positiv
- $\rightarrow$ Auffüllen mit 1, wenn Zahl negativ



Linksshift

Logische Shifts um n: Unsigned Multiplikation mit 2^n

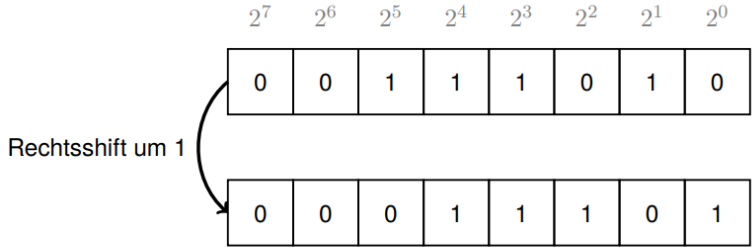
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	1	1	1	0	1	0

Multiplikation mit $4 = 2^2$

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	0	1	0	0	0

Rechtsshift

Logische Rechtsshifts um n: Unsigned Division um 2^n



$0011\_1010_2 = 58_{10}$

$0001\_1101_2 = 29_{10}$

Fragen?

Bis zum nächsten Mal ;)

Folien inspiriert von Niklas Ladurner